An Extension of SPARQL with Fuzzy Navigational Capabilities for Querying Fuzzy RDF Data

Olivier Pivert, Olfa Slama, Virginie Thion

An Extension of SPARQL with Fuzzy Navigational Capabilities for Querying Fuzzy RDF Data

Olivier Pivert, Olfa Slama, Virginie Thion
Rennes 1 University / IRISA
Lannion, France
Email: {Olivier.Pivert,Olfa.Slama,Virginie.Thion}@irisa.fr

***Abstract*—The Resource Description Framework (RDF) is the graph-based standard data model for representing semantic web information, and SPARQL is the standard query language for querying RDF data. Because of the huge volume of linked open data published on the web, these standards have aroused a large interest in the last years. This paper proposes a fuzzy extension of the SPARQL language that improves its expressiveness and usability. This extension allows (1) to query a *fuzzy RDF data model*, and (2) to express *fuzzy preferences* on data and on the *structure* of the data graph, which has not been proposed in any previous fuzzy extensions of SPARQL.**

I. Introduction

The Resource Description Framework (RDF) [25] is the standard data model promoted by the W3C for representing information about resources available on the Web. Nowadays, within the semantic web realm, the way we query RDF data is a crucial subject due to the huge quantity, the heterogenous, the vagueness, and the wide connectivity of such data. SPARQL [20], the official W3C recommendation as an RDF query language, plays the same role for the RDF data model as SQL does for relational data model. It provides basic functionalities (such as, union and optional queries, value filtering and ordering results, etc.) in order to query RDF data through graph patterns, i.e., RDF graphs containing variables data.

But classical SPARQL lacks of some expressiveness and usability capabilities as it follows a crisp (Boolean) querying of RDF data for which the response is either false or true. As a result, it lacks the ability to deal with flexibility aspects (including queries with user preferences or vagueness), which is significant in real-word applications.

From one perspective, the need to query about the structure of data and then extract relationships between resources in RDF graph, has motivated research into extending SPARQL languages to be more expressive than before. In [11], [3], [1] and [17], authors mainly extend SPARQL by allowing to query crisp RDF through graph patterns using regular expressions but they do not address the fuzziness in their approaches. From another perspective, in order to make the expression of flexible queries (involving user preferences) in RDF databases possible, [7] and [13] propose a flexible extension of SPARQL query language. The objective is to support user preferences and provide users with useful results ranked according to their preferences brought by the query. The expression of path queries is not allowed in these extensions.

There is also a real need for a flexible SPARQL that takes into account RDF graphs where data is described by intrinsic weighted values, attached to edges or nodes. This weight may denote any gradual notion like a cost, a truth value, an intensity or a membership degree. For instance, in the real world, the information stored on the Web, as well as its *metadata* are far from being perfect and are represented by vague/imprecise knowledge. An imprecise information may be of the form "An album is recent with the degree of truth 0.9".

Driven by these approaches, the RDF data model should be enriched in order to represent such information, and new query languages should be defined. Our aim in this paper is to extend SPARQL in order to support flexible querying of crisp and also fuzzy RDF graph databases (when data are imprecise).

The contribution of the paper is twofold. First, we extend the concept of SPARQL graph pattern defined over a crisp RDF data model, to the concept of fuzzy graph patterns that allows: (1) to query a *fuzzy RDF data model*, and (2) to express *fuzzy preferences* on data (through fuzzy conditions) and on the *structure* of the data graph (through fuzzy regular expressions), which has not been proposed in any previous fuzzy extensions of SPARQL. Second, we propose FURQL, an extension of the SPARQL language based the previous theoretical foundations.

The paper is organized as follows. Section II presents background notions. In Section III, which is the core of the contribution, we introduce the *fuzzy graph pattern* notion. Based on this notion, we propose the FURQL language, for which we then discuss implementation issues in Section IV. Related work is presented in Section V. At last, Section VI recalls the contributions and outlines some perspectives.

II. Background notions

The Resource Description Framework (RDF) [25] uses pairwise disjoint infinite sets of resource names, literals and blank nodes (i.e., unknown or anonymous resources) respectively denoted by $\mathcal{U}$, $\mathcal{L}$ and $\mathcal{B}$ in the following.

Let us consider an album as a resource of the Web. A characteristic may be attached to the album, like a title, an artist, a date or tracks. In order to express such a characteristic, the RDF data model uses a statement of the form of a triple $\langle \mathbf{s}, \mathbf{p}, \mathbf{o} \rangle \in (\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{L} \cup \mathcal{B})$. The subject **s** denotes the resource being described, the predicate **p** denotes the property of the resource and the object **o** denotes the property value. A triple states that the subject **s** has a property **p** with a value **o**.

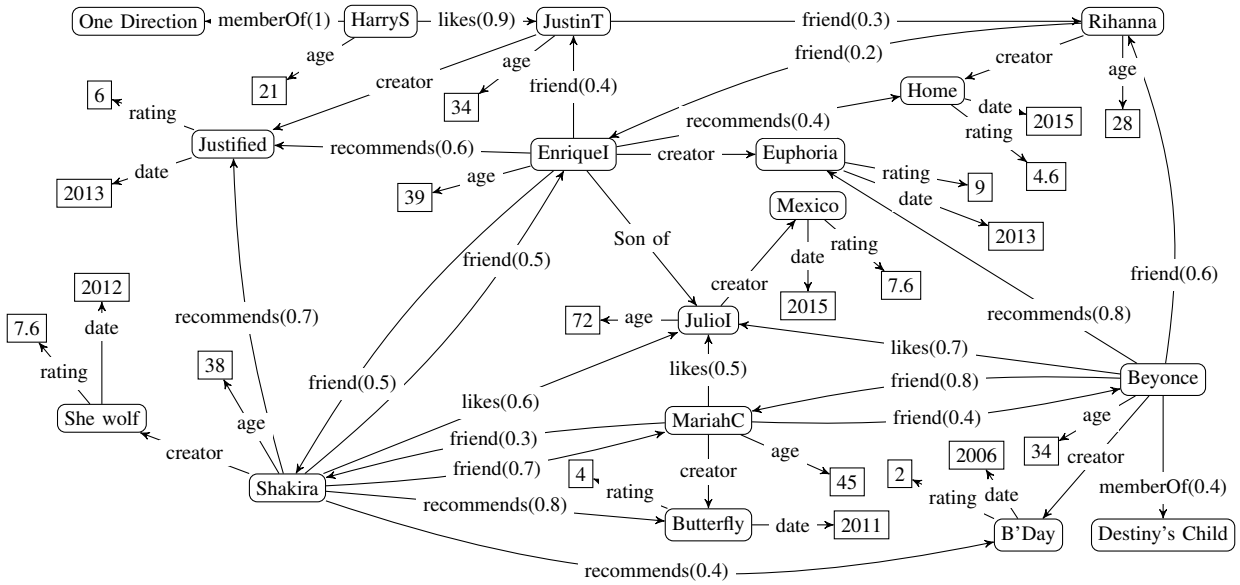


Figure 1. Fuzzy RDF graph G_{MB} inspired by MusicBrainz

For instance, the triple ⟨`Beyonce, creator, B'Day`⟩ states that `Beyonce` has `B'Day` as a `creator` property, which can be interpreted as `Beyonce` is a creator of `B'Day`.

A set of RDF triples can be modeled by a directed labeled graph (called *RDF graph* or simply *graph* in the following) where for each triple $\langle \mathbf{s}, \mathbf{p}, \mathbf{o} \rangle$, the subject **s** and the object **o** are nodes, and the predicate **p** corresponds to an edge from the subject node to the object one. RDF is then a graph-structural data model that makes it possible to exploit the basic notions of graph theory (such as node, edge, path, neighborhood, connectivity, distance, in-degree, out-degree, etc.).

RDF provides a *schema definition* language called RDF Schema (RDFS), which allows specifying semantic deductive constraints on objects, subjects and relationships of an RDF data graph. It permits to declare objects and subjects as *instances* of given classes, and *inclusion statements* between classes and between relationships. It is also possible to relate the *domain* and *range* of a relationship to classes. RDF also declares entailment rules that allow to derive new triples from the explicit triples appearing in an RDF graph. Such implicit triples are part of the RDF graph even if they do not explicitly appear in it. They can be explicitly added to the graph. When all implicit triples are made explicit in the graph then the graph is said to be *saturated*. In the following, we only consider saturated graphs.

Unfortunately, the classical crisp RDF model is only capable of representing Boolean notions whereas real-world concepts are often of a vague or gradual nature. This is why several authors proposed fuzzy extensions of the RDF model. Throughout this paper, we consider the data model based on Definition 1 which synthesizes the existing fuzzy RDF models of literature ([15], [23], [14], [12], [22], [24], [27]), whose common principle consists in adding a fuzzy degree to edges, modeled either by a value embedded in each triple or by a function associating a satisfaction degree with each triple, expressing the extent to which the fuzzy concept attached to the edge is satisfied. For instance, the fuzzy triple "(⟨`Beyonce, recommends, Euphoria`⟩, 0.8)" states that ⟨`Beyonce, recommends, Euphoria`⟩ is satisfied to the degree 0.8, which could be interpretad as `Beyonce` *strongly* recommends `Euphoria`.

Definition 1 (Fuzzy RDF (F-RDF) graph): A F-RDF graph is a tuple $(\mathcal{T}, \zeta)$ such that (i) $\mathcal{T}$ is a finite set of triples of $(\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{L} \cup \mathcal{B})$, (ii) ζ is a membership function on triples $\zeta : \mathcal{T} \rightarrow [0, 1]$.

According to the classical semantics associated with fuzzy graphs, $\zeta(t)$ qualifies the intensity of the relationship involved in the statement t. Intuitively, ζ attaches fuzzy degrees to "edges" of the graph. Having a value of 0 for ζ is equivalent to not belonging to the graph. Having a value of 1 for ζ is equivalent to fully satisfy the associated concept. In the graph G_{MB} of Figure 1, such edges appear as classical ones, i.e. with no degree attached.

The fuzzy degrees associated with edges are given or calculated. In its simplest form, each degree may be based on a simple statistical notion, e.g. the intensity of friendship between two artists may be computed as the number of their common friends over the total number of friends with respect to each artist.

Remark 1: A classical crisp RDF data graph is a special case of F-RDF data graph where the co-domains of ζ are $\{0, 1\}$. A fundamental implication is that the concepts and the flexible query language defined over a F-RDF graph in the following, remain relevant over a RDF graph.

A F-RDF graph is said to be *ground* if it contains no blank nodes. Such a graph may be ground at the beginning or made

ground e.g. by a solemnization procedure. In the following, we only consider ground graphs.

Example 1 (Fuzzy RDF graph): Figure 1 is an example of Fuzzy RDF graph inspired by MusicBrainz[1]. This graph, denoted G_{MB} in the following, is the running example of the article. It mainly contains artists and albums as nodes. For readability reasons, each URI node contains the value of its *name* instead of the URI itself. Literal values may be attached to URI, like the age of an artist, the release date or the global rating of an album. The graph contains fuzzy relationships (e.g. friend, likes, recommends, memberOf) as well as crisp ones (e.g. creator, date,...). We limit our example to some entities including artists and albums and omit URI prefixes to avoid overcrowding the figure.

In order to create this graph, we started from a MusicBrainz nonfuzzy subgraph for which every relationship between nodes is Boolean and, then, we made it fuzzy by adding satisfaction degrees denoting intensity on some relationships. Here for instance, the degree associated with an edge of the form Art $-$ memberOf $\rightarrow$ Group is the number of years the artist stayed in this group over the number of years this group has been existing. $\diamond$

In the following, we rely on classical notions from fuzzy graph theory [21], which are the *path*, the *distance* and the *strength* (ST) of the connection between two nodes.

Let G be a F-RDF data graph. Classically, a *path* p in G is denoted by a possibly empty sequence of triples $(t_1, \cdots, t_k, \cdots, t_n)$ such that $\{t_i \mid 1 \leq i \leq n\} \subseteq G$ and for all $1 \leq k \leq n-1$, the object of t_k is the subject of t_{k+1}.

Given two nodes x and y, $Paths(x, y)$ denotes the set of cycle-free paths[2] in G connecting x to y (the set of paths of the form $(t_1, \cdots, t_k, \cdots, t_n)$ such that x is the subject of t_1 and y is the object of t_n).

The *distance* between two nodes x and y is defined by

$$distance(x, y) = min_{p \in Paths(x,y)} length(p) \qquad (1)$$

where $length(p)$ is the length of a path p in a fuzzy graph [21], calculated by $length(p) = \sum_{t \in p} \frac{1}{\zeta(t)}$.

The *strength* between two nodes x and y is defined by

$$ST(x, y) = max_{p \in Paths(x,y)} ST\_path(p) \qquad (2)$$

where $ST\_path(p)$ is the strength of the path connecting x and y in a fuzzy graph [21], calculated by $ST\_path(p) = \min(\{\zeta(t) | t \in p\}$.

Example 2: Let us consider the cycle-free paths from G_{MB} connecting Beyonce to Euphoria, depicted in Figure 2, and let us compute the *distance* and the *strength* between the pair of nodes *(Beyonce, Euphoria)*. The distance between the pair of nodes *(Beyonce, Euphoria)* is calculated as *distance(Beyonce, Euphoria) = min (length(p_1), length(p_2), length(p_3))*, with *length (p_1)= 1/0.8= 1.25*, *length(p_2) = 1/0.6 + 1/0.2 + 1= 7.7)* and *length (p_3)= 1/0.8 + 1/0.3 + 1/0.5 + 1 = 7.5*. Finally, the length of the shortest path between the pair of nodes *(Beyonce, Euphoria)* is *distance(Beyonce, Euphoria)= 1.25*. The strength between the pair of nodes *(Beyonce, Euphoria)* is calculated as *ST(Beyonce, Euphoria) = max (ST_path(p_1), ST_path(p_2), ST_path(p_3))*, with *ST_path(p_1)= 0.8*, *ST_path(p_2)= 0.2* and *ST_path(p_3)= 0.3*. Thus, *ST(Beyonce, Euphoria)= 0.8*. Here, the distance and the strength correspond to the same path, but it is of course not necessarily the case in general. $\diamond$

[1]https://musicbrainz.org/

[2]Considering paths containing cycle would not change the result of the following expressions (1) and (2).

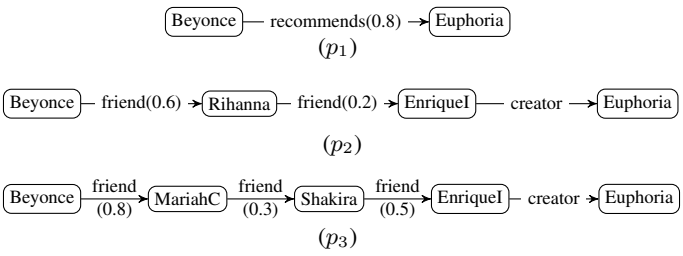


Figure 2. Cycle-free paths from G_{MB} connecting Beyonce to Euphoria

SPARQL [20] is the standard query language promoted by the W3C for querying RDF Data. It is a declarative query language based on graph pattern matching, in the sense that the query processor searches for sets of triples in the data graph that satisfy a pattern (i.e., set of triples containing variables) expressed in the query. A SPARQL query has the general form given in Listing 1, where the clause `SELECT` is for specifying which variables should be returned, the clause `FROM` defines the datasets to be queried, and the clause `WHERE` contains the triple of the researched pattern.

```
SELECT ... #Result
FROM ...   #Dataset definition
WHERE ...  #Graph pattern
ORDER BY ..., DISTINCT ..., ... #Modifiers
```

Listing 1. Skeleton of a SPARQL query

Roughly speaking, a graph pattern is defined as being triples where variables can occur, composed by binary operators `UNION`, `FILTER`, `OPTIONAL` and . (concatenation). Listing 2 gives an example of SPARQL query that retrieves the albums of 2012 that are either created or liked by Shakira, with the associated rating if it is available.

```
SELECT ?album ?r WHERE
{
  { ?artist name "Shakira".
    ?artist creator ?album. }
UNION
  { ?artist name "Shakira".
    ?artist likes ?album. }
OPTIONAL { ?album rating ?r. }
?album date ?d.
FILTER (?d = "2012").
}
```

Listing 2. A SPARQL query

SPARQL also provides solution modifiers, which make it possible to modify the result set by applying classical operators like `ORDER BY`, `DISTINCT`, `LIMIT`, `PROJECTION`, or `OFFSET`.

In the following section, we introduce the notion of *fuzzy graph pattern*, which is a fuzzy extension of the *graph pattern* notion introduced in [16] and [4]. A fuzzy graph pattern allows to express fuzzy preferences on the entities of an F-RDF graph (through fuzzy conditions) and on the structure of the graph (through fuzzy regular expressions).

III. Fuzzy graph patterns

The introduced foundations rely on the SPARQL graph pattern syntax and semantics introduced in [16] and [4], which present SPARQL graph patterns in a more traditional algebraic formalism than the official syntax does [25]. It considers the following binary operators: AND (SPARQL `concatenation`), UNION (SPARQL `UNION`), OPT (SPARQL `OPTIONAL`) and FILTER (SPARQL `FILTER`). We redefine the associated syntax and semantics in order to introduce fuzzy conditions and fuzzy regular expressions expressed over the F-RDF data model of Definition 1.

Syntax of fuzzy graph patterns

For the following, we assume the existence of an infinite set $\mathcal{V}$ of variables such that $\mathcal{V} \cap (\mathcal{U} \cup \mathcal{L}) = \emptyset$. By convention, we prefix the elements of $\mathcal{V}$ by a question mark symbol. Before giving the formal definition of a fuzzy graph pattern, we introduce the notion of a *fuzzy regular expression*.

Definition 2 (Fuzzy regular expression): The set $\mathcal{F}$ of fuzzy regular expression patterns, defined over the set $\mathcal{U}$ of URIs, is recursively defined by:

- ϵ is a fuzzy regular expression of $\mathcal{F}$ denoting the empty pattern;
- $u \in \mathcal{U}$ and '_' are a fuzzy regular expressions of $\mathcal{F}$;
- if $A \in \mathcal{F}$ and $B \in \mathcal{F}$ then $A|B, A.B, A^*, A^{cond}$ are fuzzy regular expressions of $\mathcal{F}$.

The character '_' denotes any element of $\mathcal{U}$, $A|B$ denotes alternative expressions, $A.B$ denotes the concatenation of expressions, A^* stands for the classical repetition of an expression (the Kleene closure), A^{cond} denotes paths satisfying the pattern A with a condition $cond$ where $cond$ is a Boolean combination of atomic formulas of the form: $sprop$ IS $Fterm$ where $sprop$ is a structural property of the path defined by the expression and $Fterm$ denotes a predefined or user-defined fuzzy term like $short$ (see figure 3). In the following, we limit the path structural properties to ST and $distance$ (see Section II). We denote by A^+ the classical shortcut for $A.A^*$.

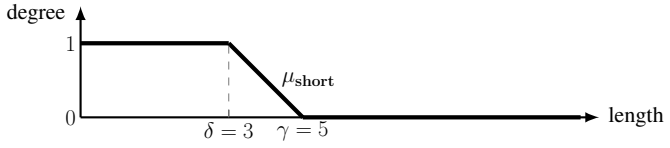


Figure 3. Representation of the fuzzy term *short*

Definition 3 (fuzzy graph pattern): Fuzzy graph patterns are recursively defined by:

- A fuzzy triple from $(\mathcal{U} \cup \mathcal{V}) \times (\mathcal{U} \cup \mathcal{F} \cup \mathcal{V}) \times (\mathcal{U} \cup \mathcal{L} \cup \mathcal{V})$ is a fuzzy graph pattern.
- If P_1 and P_2 are fuzzy graph patterns then (P_1 AND P_2), (P_1 UNION P_2) and (P_1 OPT P_2) are fuzzy graph patterns.
- If P is a fuzzy graph pattern and C is a *fuzzy condition* then (P_1 FILTER C) is a fuzzy graph pattern. A *fuzzy condition* is a logical combination of fuzzy terms defined by:
  - if $\{?x, ?y\} \subseteq \mathcal{V}$ and $c \in (\mathcal{U} \cup \mathcal{L})$, then $bound(?x)$, $?x\ \theta\ c$ and $?x\ \theta\ ?y$ are fuzzy conditions, where θ is a fuzzy or crisp comparator,
  - if $?x \in \mathcal{V}$ and $Fterm$ is a fuzzy term then, $?x$ IS $Fterm$ is a fuzzy condition,
  - if C_1 and C_2 are fuzzy conditions then $(\neg C_1)$ and $(C_1 \odot C_2)$ (where $\odot$ is a fuzzy connective) are fuzzy conditions. Fuzzy connectives include of course fuzzy conjunction $\wedge$ (resp. disjunction $\vee$), usually interpreted by the triangular norm minimum (resp. maximum), but also many other operators that may be used for expressing different kinds of trade-offs, such as the weighted conjunction and disjunction [8], mean operators, fuzzy quantifiers [9], or the non-commutative connectives described in [6].

Given a pattern P, $var(P)$ denotes the set of variables occurring in P.

Semantics of fuzzy graph patterns

Intuitively, given an F-RDF data graph G, the semantics of a fuzzy graph pattern P defines a set of mappings, where each mapping (from $var(P)$ to URIs and literals of G) maps the pattern to an isomorphic subgraph of G. For introducing such a concept, the notion of satisfaction of a fuzzy regular expression must first be defined.

Definition 4 (Fuzzy regular expression matching of a path): Let $G = (\mathcal{T}, \zeta)$ be an F-RDF graph and exp be a fuzzy regular expression. Let $p = (\langle s_1, p_1, o_1\rangle, ..., \langle s_n, p_n, o_n\rangle) \subseteq G$ be a path of G. *p satisfies exp with a satisfaction degree of* $sat_{exp}(p)$ is defined as follows, according to the form of exp (in the following, f, f_1 and f_2 are fuzzy regular expressions):

- exp is of the form ϵ. If p is empty then $sat_{exp}(p) = 1$ else $sat_{exp}(p) = 0$.
- exp is of the form $u \in \mathcal{U}$ (resp. "_"). If p_1 is u (resp. any $u \in \mathcal{U}$) then $sat_{exp}(p) = \zeta(\langle s_1, p_1, o_1\rangle)$ else 0.
- exp is of the form $f_1.f_2$. Let P be the set of all pairs of paths (p_1, p_2) s.t. p is of the form p_1p_2. One has $sat_{exp}(p) = \max_P(\min(sat_{f_1}(p_1), sat_{f_2}(p_2)))$.
- exp is of the form $f_1 \cup f_2$. One has $sat_{exp}(p) = \max(sat_{f_1}(p), sat_{f_2}(p))$.
- exp is of the form f^*. If p is the empty path then $sat_{exp}(p) = 1$. Otherwise, we denote by P the set of all tuples of paths $(p_1, \cdots, p_n)$ $(n > 0)$ s.t. p is of the form $p_1 \cdots p_n$. One has $sat_{exp}(p) = \max_P(\min_{i\in[1..n]}(sat_{exp}(p_1)))$.
- exp is of the form f^{Cond} where $Cond$ where $cond$ is a fuzzy condition. $sat_{exp}(p) = \min(sat_f(p),\ \mu_{Cond}(p))$ where $\mu_{Cond}(p)$ denotes the degree of satisfaction of $Cond$ by p.

Again, not satisfying is equivalent to getting a degree of 0.

Definition 5 (Satisfaction of a fuzzy regular expression of a pair of nodes): Let $G = (\mathcal{T}, \zeta)$ be an F-RDF graph and exp be a fuzzy regular expression. Let (x, y) be a pair of nodes of G. (x, y) *satisfies* exp *with a satisfaction degree of* $sat_{exp}(x, y)$ is defined by $max_{p \in Paths(x,y)} sat_{exp}(p)$.
Note that only cycle-free paths need to be considered in order to compute the satisfaction degree.

Example 3 (Satisfaction of a fuzzy regular expression): Let us consider the following fuzzy regular expressions, for which we give the semantics and their satisfaction according to G_{MB}. Note that the paths represented in Figure 4 are some cycle-free paths among many others from the graph G_{MB}.

Expression $f_1 = (\textit{friend}^+).\textit{creator}$ is a fuzzy regular expression. A pair of nodes (x, y) satisfies f_1 if x has a "friend-linked" artist (an artist connected to x with a path made of *friend* edges), that created the album y. All of the pairs of nodes *(EnriqueI, Justified)*, *(Shakira, Butterfly)*, *(Beyonce, Euphoria)*, *(MariahC, Euphoria)* and *(Shakira, Euphoria)*, illustrated in Figure 4, satisfy f_1 with the satisfaction degree sat_{f_1}*(EnriqueI, Justified) = 0.4*, sat_{f_1}*(Shakira, Butterfly) = 0.7*, sat_{f_1}*(Beyonce, Euphoria) = 0.3*, sat_{f_1}*(MariahC, Euphoria) = 0.3* and sat_{f_1}*(Shakira, Euphoria) = 0.5* respectively.

Expression $f_2 = (\textit{friend}^+)^{distance\ is\ short}.\textit{creator}$ is a fuzzy regular expression. A pair of nodes (x, z) satisfies f_2 if x has a "close" friend artist y that created an album z, "close" meaning that x is connected to y by a *short* path made of *friend* edges (see Figure 3). According to Figure 4, the pairs of nodes *(EnriqueI, Justified)*, *(Shakira, Butterfly)* and *(Shakira, Euphoria)* are the only ones that match f_2 as $\mu_{short}(3.5) = 0.75$ (where *length* of pair *(EnriqueI, Justified)* = 1/0.4 + 1 = 3.5), $\mu_{short}(2.4) = 1$ (where *length* of pair *(Shakira, Butterfly)*= $1/0.7 + 1 = 2.4$), $\mu_{short}(7.7) = 0$ (where *length* of pair *(Beyonce, Euphoria)* = 1/0.6 + 1/0.2 + 1 = 7.7), $\mu_{short}(6.33) = 0$ (where $length$ of pair *(MariahC, Euphoria)*= 1/0.3 + 1/0.5 + 1= 6.33) and $\mu_{short}(3) = 1$ (where *length* of pair *(Shakira, Euphoria)* = 1/0.5 + 1 = 3) with satisfaction degree of sat_{f_2}*(EnriqueI, Justified) = 0.4*, sat_{f_2}*(Shakira, Butterfly) = 0.7* and sat_{f_2}*(Shakira, Euphoria) = 0.5* respectively. ⋄

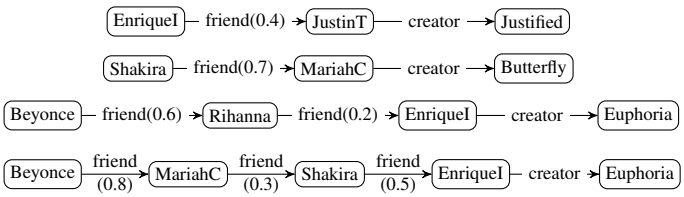


Figure 4. Some paths from G_{MB}.

Let us now come to the definition of a mapping. A mapping is a pair (m, d) where $m : \mathcal{V} \to (\mathcal{U} \times \mathcal{L})$ and $d \in [0, 1]$. Intuitively, m maps the variables of a fuzzy graph pattern into a subgraph ("answer") of the F-RDF data graph and d denotes the satisfaction degree associated with the mapping (the more satisfactory the subgraph, the higher the satisfaction degree). $m(t)$, where t is a triple pattern, denotes the triple obtained by replacing each variable x of t by $m(x)$. The domain of a mapping m denoted by $dom(m)$ is the subset of $\mathcal{V}$ for which m is defined. Two mappings m_1 and m_2 are compatible iff for all $?v \in dom(m_1) \cap dom(m_2)$, one has $m_1(?v) = m_2(?v)$. Intuitively, m_1 and m_2 are compatible if m_1 can be extended with m_2 to obtain a new mapping $m_1 \oplus m_2$ and vice versa.

Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two sets of mappings. We define the join, union, difference and left outer-join of $\mathcal{M}_1$ with $\mathcal{M}_2$ as:

$\mathcal{M}_1 \bowtie \mathcal{M}_2 = \{(m_1 \oplus m_2, \min(d_1, d_2)) \mid (m_1,\ d_1) \in \mathcal{M}_1$ and $(m_2,\ d_2) \in \mathcal{M}_2$ and $m_1,\ m_2$ are compatible$\}$.

The operation $\mathcal{M}_1 \bowtie \mathcal{M}_2$ denotes the set of new mappings that result from extending mappings in $\mathcal{M}_1$ with their compatible mappings in $\mathcal{M}_2$,

$\mathcal{M}_1 \cup \mathcal{M}_2 =$
$\{(m,\ d) \mid (m,\ d) \in \mathcal{M}_1$ and $m \notin support(\mathcal{M}_2)\} \cup$
$\{(m,\ d) \mid (m,\ d) \in \mathcal{M}_2$ and $m \notin support(\mathcal{M}_1)\} \cup$
$\{(m,\ \max(d_1, d_2)) \mid (m,\ d_1) \in \mathcal{M}_1$ and $(m,\ d_2) \in \mathcal{M}_2\}$

Here, $\cup$ corresponds to the classical set-theoretic union.

$\mathcal{M}_1 \backslash \mathcal{M}_2 = \{(m_1,\ d_1) \mid\ (m_1,\ d_1) \in \mathcal{M}_1$ and $\forall (m_2,\ d_2) \in \mathcal{M}_2$, m_1 and m_2 are not compatible$\}$.

$\mathcal{M}_1 \backslash \mathcal{M}_2$ returns the set of mappings in $\mathcal{M}_1$ that cannot be extended with any mapping in $\mathcal{M}_2$.

$\mathcal{M}_1 \rtimes \mathcal{M}_2 = (\mathcal{M}_1 \bowtie \mathcal{M}_2) \cup (\mathcal{M}_1 \backslash \mathcal{M}_2)$.

A mapping m is in $\mathcal{M}_1 \rtimes \mathcal{M}_2$ if it is the extension of a mapping of $\mathcal{M}_1$ with a compatible mapping of $\mathcal{M}_2$, or if it is in $\mathcal{M}_1$ and cannot be extended with any mapping of $\mathcal{M}_2$.

Definition 6 (Mapping satisfying a fuzzy condition): Let m be a mapping and C be a fuzzy condition. Then m satisfies the fuzzy condition C with a satisfaction degree defined as follows, according to the form of C:

- C is of the form $bound(?x)$: if $?x \in dom(m)$ then m satisfies the condition C with a degree of 1, else 0.
- C is of the form $?x\,\theta\,c$: if $?x \in dom(m)$ then m satisfies the condition C with a degree of $\mu_\theta(m(?x), c)$, else 0.
- C is of the form $?x\ \theta\ ?y$: if $?x \in dom(m)$ and $?y \in dom(m)$, then m satisfies the condition C with a degree of $\mu_\theta(m(?x), m(?y))$, else 0.
- C is of the form $?x$ IS $Fterm$: if $?x \in dom(m)$ then m satisfies the condition C to the degree $\mu_{Fterm}(m(?x))$ (which can be 0).
- C is of the form $\neg C_1$ or $C_1 \odot C_2$ where $\odot$ is a fuzzy connective: we use the usual interpretation of the fuzzy operator involved (complementation to 1 for the negation, minimum for the conjunction, maximum for the disjunction, etc (see e.g. [9]).

Definition 7 (Evaluation (interpretation) of a fuzzy graph pattern): The evaluation of a fuzzy graph pattern P over an F-RDF graph, denoted by $[\![P]\!]_G$ is recursively defined by:

- if P is of the form of a triple graph pattern $t \in (\mathcal{U} \cup \mathcal{V}) \times (\mathcal{U} \cup \mathcal{V}) \times (\mathcal{U} \times \mathcal{L} \times \mathcal{V})$ then $[\![P]\!]_G = \{(m, 1) \mid dom(m) = var(t)$ and $m(t) \in G\}$,

- if P is of the form of a fuzzy triple graph pattern $t \in (\mathcal{U} \cup \mathcal{V}) \times \mathcal{F} \times (\mathcal{U} \times \mathcal{L} \times \mathcal{V})$ denoted by $\langle ?x, exp, ?y \rangle$ (where variables occur as subject and object) then $[\![P]\!]_G = \{(m, d) \mid dom(m) = \{?x, ?y\}$ and $(m(?x), m(?y))$ satisfies exp with a satisfaction degree $d_{exp}(x, y)\}$. The case for which the subject (resp. the object) of t is a constant of $\mathcal{U}$ (resp. $\mathcal{U} \cup \mathcal{L}$) is trivially induced from this definition.
- if P is of the form P_1 AND P_2 then $[\![P]\!]_G = [\![P_1]\!]_G \bowtie [\![P_2]\!]_G$,
- if P is of the form P_1 OPT P_2 then $[\![P]\!]_G = [\![P_1]\!]_G \mathbin{⟕} [\![P_2]\!]_G$,
- if P is of the form P_1 UNION P_2 then $[\![P]\!]_G = [\![P_1]\!]_G \cup [\![P_2]\!]_G$,
- if P is of the form P_1 FILTER C then $[\![P]\!]_G = \{(m, d) \mid m \in [\![P]\!]_G$ and m satisfies C to the degree of $d\}$.

Intuitively, expressions $(P_1$ AND $P_2)$, $(P_1$ UNION $P_2)$, $(P_1$ OPT $P_2)$, and $(P_1$ FILTER $C)$ refer to conjunction graph patterns, union graph patterns, optional graph patterns, and filter graph patterns respectively. Optional graph pattern allows a partial match for the query (i.e. the query tries to match a graph patterns and does not omit solution where some part of the optional patterns does not satisfy this query).

Note that a crisp graph pattern is a special case of fuzzy graph pattern where no fuzzy term or condition occurs.

Example 4 (Evaluation of a fuzzy graph pattern): Let us consider P_{rec_low} the fuzzy graph pattern defined by *(?Art1,* $(friend^+)^{distance\ is\ short}$*.creator, ?Alb) AND (?Art1, recommends, ?Alb) AND ((?Alb, rating, ?r) FILTER (?r is low))*, for which Figure 6 is a graphical representation.

Intuitively, P_{rec_low} retrieves the list of artists *(?Art1)* in G_{MB} s.t. *(?Art1)* recommends a low rated album *(?Alb)* created by another artist who is a close friend of them *(?Art1)*.

Figure 7 gives the set of subgraphs of G_{MB} satisfying the pattern P_{rec_low}. The matching value of *Art1* is either *Shakira* or *EnriqueI* who match the pattern P_{rec_low} (i.e they are the only artists that have liked a low rated album created by another artist among their close friends). Note that $(friend^+)^{distance\ is\ short}$*.creator* is the fuzzy regular expression f_2 of Example 3 with sat_{f_2}*(EnriqueI, Justified) = 0.4*, sat_{f_2}*(Shakira, Butterfly) = 0.7* and sat_{f_2}*(Shakira, Euphoria) = 0.5* and we consider μ_{low_rating}*(4) = 0.66*, μ_{low_rating}*(6) = 0.33* and μ_{low_rating}*(9) = 0* defined in Figure 5.

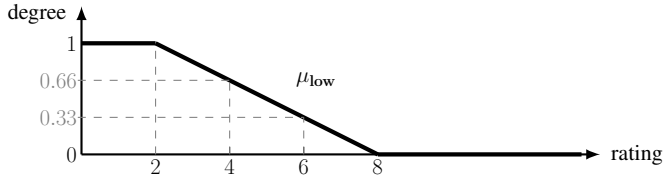


Figure 5. Representation of the fuzzy term *low* applied to a *rating* value

Then, the evaluation of the pattern P_{rec_low} over the RDF graph G_{MB} includes two mappings with their respective satisfaction degrees and is represented as follows:

$[\![P_{rec\_low}]\!]_{G_{MB}}$ = {({*?Art1* → *EnriqueI* , *?Alb* → *Justified, ?r* → *6*}, *0.33*), {*?Art1* → *Shakira* , *?Alb* → *Butterfly, ?r* → *4*}, *0.66*)}. ⋄

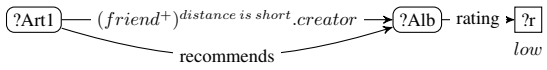


Figure 6. Graphical representation of pattern P_{rec_low}

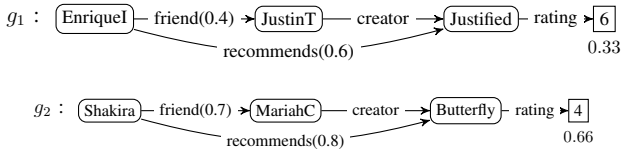


Figure 7. Subgraphs satisfying P_{rec_low}

IV. About the FURQL query language and implementation issues

We now introduce FURQL (Fuzzy RDF Query Language), which consists in extending SPARQL graph patterns by fuzzy graph patterns. We can only outline the syntax of the language here, due to space limitation.

Syntactically the extension naturally extends the SPARQL one, by allowing the occurrence of fuzzy graph patterns in the `WHERE` clause and the occurrence of fuzzy conditions in the `FILTER` clause. A fuzzy regular expression is close to a property path, as defined in SPARQL 1.1 [10], and involves a fuzzy structural property (e.g. *distance* and *strength* over fuzzy graphs). Semantically, the extension relies on the semantics defined in Section III.

Example 5: The FURQL query of Listing 3 retrieves artists that recommend low-rated albums by close friends (see Pattern P_{rec_low} of Example 4), and performs an alpha-cut on the answers (only those having a satisfaction degree greater or equal to 0.4 are kept). The `CUT` clause is of course optional.

```
SELECT ?art1 WHERE
{
 { ?art1 (friend+ | distance IS short) ?art2.
   ?art2 creator ?alb.
   ?alb rating ?r.
   ?art1 recommends ?alb. }
FILTER (?r IS low).
} CUT 0.4
```

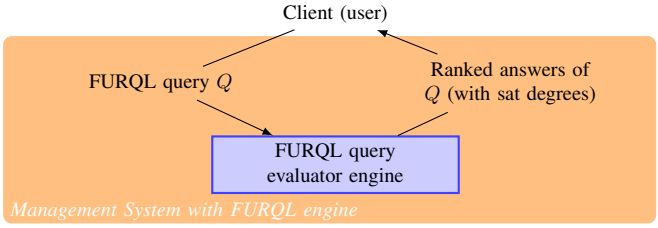
Listing 3. A FURQL query containing P_{rec_low}

The result of this query over G_{MB} is the singleton {EnriqueI} which is $m(?art1)$ in the mapping {?art1 → EnriqueI , ?alb → Justified, ?r → 6}, i.e., the only mapping of $[\![P_{rec\_low}]\!]_{G_{MB}}$ having a satisfaction degree greater or equal to 0.4 (see Example 4). ⋄

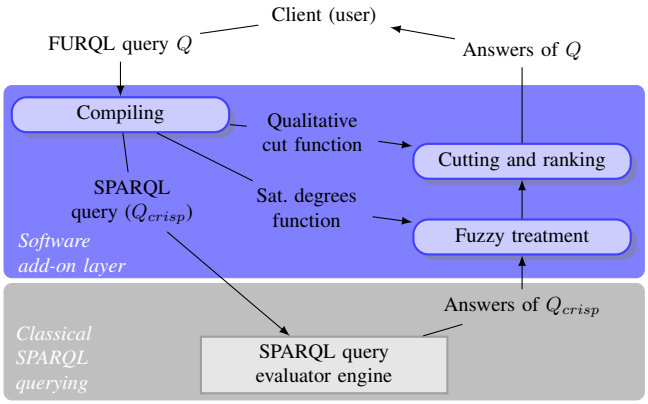
We now discuss implementation issues related to our proposal. Two aspects have to be considered: i) the storage of fuzzy RDF graphs, and ii) the evaluation of FURQL queries.

The first point does not raise any serious problem since one may use the *reification mechanism* that makes it possible to attach fuzzy degrees to triples, as proposed in [22].

Concerning the evaluation of FURQL queries, two architectures may be thought of. A first solution consists in implementing a specific query evaluation engine inside the data management system. Figure 8.(a) is an illustration of this architecture. The advantage of this solution is that optimization techniques implemented directly in the query engine should make the system very efficient for query processing. An important downside is that the implementation effort is substantial, but the strongest objection for this solution is that the evaluation of a FURQL query in a distributed architecture would imply having available a FURQL query evaluator at each SPARQL endpoint, which is not realistic at the time being. An alternative, more realistic architecture consists in adding a FURQL software layer over a standard — and possibly distant — classical SPARQL engine (endpoint). This layer basically consists in a *query compiler* producing a (crisp) SPARQL query for retrieving the appropriate data, a (query-dependent) function for computing the satisfaction degrees, and a (query-dependent) function for ranking the answers (and filtering them in case an α-cut has to be returned). Figure 8.(b) is an illustration of this architecture. Deriving a SPARQL query from a FURQL query should rely on the *derivation principle* proposed in [5] in a relational database context and applied in [13] to "simple" fuzzy SPARQL queries (i.e., queries that do not involve any structural conditions). It remains to be studied how the different types of structural conditions allowed in FURQL can be derived into crisp conditions in the syntax of SPARQL 1.1., but note that it is always possible to derive them into *true* (which amounts to discarding them from the crisp SPARQL query) and to take them into account only in the "fuzzy treatment" module (cf. Figure 8.(b)) if no better derivation can be found. This solution may look extreme at first sight but it could be in fact reasonable if the non-structural part (i.e., the "value-based" part) of the global fuzzy condition in the query is selective enough for avoiding a plethoric set of answers to assess.



(a) Implement a specific FURQL query evaluation engine



(b) Adding a software layer

Figure 8. Possible architectures for implementing the FURQL language

V. Related work

Related work concerns two categories of approaches: those that extend SPARQL with navigational capabilities, and those that extend it with fuzzy querying functionalities. To the best of our knowledge, there does not exist any work in the literature that does both.

In the first category, some new query languages [11], [3], [17], [1] have been designed to overcome the limited functionalities of the standard query language SPARQL in terms of path extractions queries (generally of unknown length) within RDF databases. These languages are more expressive and extend SPARQL by allowing querying RDF data through graph path patterns by using regular expressions. Motivated by this concern, [11] and [3] introduce respectively SPARQLeR and SPARQ2L, two extensions of the classical query language SPARQL with path patterns that represent (undirected and directed) paths between nodes in the RDF graph. They both extend the SPARQL graph patterns with path variables (i.e., path triple patterns) in the predicate position. This makes it possible to express more complex queries such as subgraph extraction queries (e.g. how are A, B, C and D related?), reachability queries (e.g. does there exist a path from A to F), etc. More recently, [1] proposed an extension of SPARQL which captures more expressive graph patterns, called PSPARQL (Path SPARQL). This query language is a combination of SPARQL and graph query languages (namely G, G+ and Graphlog). The authors propose a syntactic and semantic redefinition of the RDF model making it possible to express regular expression patterns instead of predicates in the RDF triple. This new model is called Path RDF (PRDF) which is an extension of a GRDF graph (i.e., it permits the expression of variables as predicates, as well as literals as subjects) with regular expression patterns constructed over the set of urirefs and the set of variables in the predicate position. Then, they develop the query language PSPARQL, which extends SPARQL with PRDF graph patterns (i.e., graph patterns with (involving) regular expression) able to find nonsimple and arbitrary length path queries. In [2], the same authors extend (P)SPARQL by adding the ability to express complex constraints over regular expressions during path search. Another important related work — that we chose as one of the bases of our own approach as it is very well formalized — is [17] where the authors introduce a new query language named nSPARQL, which makes it possible

to express complex navigation statements in an RDF graph. With nSPARQL, one may navigate through a ground RDF graph using the notion of nested regular expressions in the predicate position.

As for the second category — that gathers approaches that deal with the fuzzy querying issue —, different researchers have extended SPARQL so as to make it more flexible. A variety of proposals aimed to introduce preferences into SPARQL queries can be found in the literature (see the survey paper [18]). These approaches may be classified into two categories: i) quantitative ones (fuzzy-set-based and top-k queries, and ii) qualitative ones that extend the classical Skyline approach. To the best of our knowledge, the existing fuzzy-set-based approaches, namely [7], [26], [13], make it possible to express preferences on the value of the nodes but *not* on the structure of the RDF graph, contrary to ours. This is also the case, by the way, of the other flexible querying approaches, either qualitative or quantitative, see [18].

Let us mention, however, that the expression of fuzzy preferences involving both a value-based and a structural aspect was first tackled in [19] in a more general context than that of RDF databases, namely that of *graph databases*. The present work can thus be seen as a (nontrivial) adaptation of the concepts from [19] to the RDF/SPARQL setting.

VI. Conclusion

In this paper, we have defined a fuzzy extension of SPARQL that goes beyond the previous proposals in terms of expressiveness inasmuch as it makes it possible i) to deal with *fuzzy RDF data*, and ii) to express fuzzy *structural* conditions beside more classical fuzzy conditions on the values of the nodes present in the graph. The language, called FURQL, is based on the notion of *fuzzy graph pattern* which extends Boolean graph patterns introduced by several authors in a crisp querying context.

Perspectives for future work are manifold. First, it is of course necessary to deal more in-depth with implementation issues (which could only be briefly discussed here) and to implement a prototype illustrating the power of the approach on real-world RDF databases. Secondly, different possibilities exist for extending the language with more sophisticated fuzzy conditions. One may think for instance of conditions involving fuzzy quantifiers, and of fuzzy predicates related to some structural properties of the nodes (centrality, in-degree, out-degree, etc). It would also be worth investigating the way this framework could be applied to the management of quality-related metadata (about freshness, reliability and so on) which are in general of a fuzzy nature.

Acknowledgement: This work has been partially funded by the French DGE (Direction Générale des Entreprises) under the project ODIN (Open Data INtelligence).

References

[1] F. Alkhateeb, J.-F. Baget, and J. Euzenat. Extending SPARQL with regular expression patterns (for querying RDF). *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(2):57–73, 2009.

[2] F. Alkhateeb and J. Euzenat. Constrained regular expressions for answering RDF-path queries modulo RDFS. *Intl. Journal of Web Information Systems*, 10(1):24–50, 2014.

[3] K. Anyanwu, A. Maduko, and A. Sheth. SPARQ2L: towards support for subgraph extraction queries in RDF databases. In *Proc. of the Intl. conference on World Wide Web*, pages 797–806. ACM, 2007.

[4] M. Arenas and J. Pérez. Querying semantic web data with SPARQL. In *Proc. of ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 305–316, 2011.

[5] P. Bosc and O. Pivert. SQLf query functionality on top of a regular relational database management system. In *Knowl. Mgmt. in Fuzzy Databases*, pages 171–190. Springer, 2000.

[6] P. Bosc and O. Pivert. On four noncommutative fuzzy connectives and their axiomatization. *Fuzzy Sets and Systems*, 202:42–60, 2012.

[7] J. Cheng, Z. Ma, and L. Yan. f-SPARQL: a flexible extension of SPARQL. In *Proc. of the Intl. Conf. on Database and Expert Systems Applications (DEXA)*, pages 487–494, 2010.

[8] D. Dubois and H. Prade. Weighted minimum and maximum operations in fuzzy set theory. *Inf. Sci.*, 39(2):205–210, 1986.

[9] J. Fodor and R. Yager. Fuzzy-set theoretic operators and quantifiers. In D. Dubois and H. Prade, editors, *The Handbooks of Fuzzy Sets Series, vol. 1: Fundamentals of Fuzzy Sets*, pages 125–193. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2000.

[10] S. Harris and A. Seaborne. SPARQL 1.1 Query Language. W3C Recommendation, 2013. http://www.w3.org/TR/sparql11-query.

[11] K. J. Kochut and M. Janik. Sparqler: Extended sparql for semantic association discovery. In *The Semantic Web: Research and Applications*, pages 145–159. Springer, 2007.

[12] Y. Lv, Z. Ma, and L. Yan. Fuzzy RDF: A data model to represent fuzzy metadata. In *Fuzzy Systems, 2008. FUZZ-IEEE 2008. (IEEE World Congress on Computational Intelligence). IEEE Intl. Conf. on*, pages 1439–1445, June 2008.

[13] R. Ma, X. Jia, J. Cheng, and R. Angryk. SPARQL queries on RDF with fuzzy constraints and preferences. *Journal of Intelligent & Fuzzy Systems*, 202:1–13, 2015.

[14] M. Mazzieri and A. Dragoni. A fuzzy semantics for the resource description framework. In P. da Costa, C. dAmato, N. Fanizzi, K. Laskey, K. Laskey, T. Lukasiewicz, M. Nickles, and M. Pool, editors, *Uncertainty Reasoning for the Semantic Web I*, volume 5327 of *Lecture Notes in Computer Science*, pages 244–261. Springer Berlin Heidelberg, 2008.

[15] M. Mazzieri, A. F. Dragoni, and U. P. D. Marche. A fuzzy semantics for semantic web languages. In *Proc. of Workshop URSW at the Intl. Semantic Web Conf. (ISWC)*, pages 12–22, 2005.

[16] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3):16:1–16:45, Sept. 2009.

[17] J. Pérez, M. Arenas, and C. Gutierrez. nSPARQL: A navigational language for RDF. *J. Web Sem.*, 8(4):255–270, 2010.

[18] O. Pivert, O. Slama, and V. Thion. SPARQL extensions with preferences: a survey. In *Proc. of the ACM Symposium on Applied Computing*, SAC, 2016.

[19] O. Pivert, V. Thion, H. Jaudoin, and G. Smits. On a fuzzy algebra for querying graph databases. In *Tools with Artificial Intelligence (ICTAI), 2014 IEEE 26th Intl. Conf. on*, pages 748–755. IEEE, 2014.

[20] E. Prud and A. Seaborne. SPARQL query language for RDF. W3C Recomm., 2008.

[21] A. Rosenfeid. Fuzzy graphs. In *Proc. of the US–Japan Seminar on Fuzzy Sets and Their Applications*, page 77. Academic Press, 2014.

[22] U. Straccia. A Minimal Deductive System for General Fuzzy RDF. In *Proc of the Intl. Conf on Web Reasoning and Rule Systems (RR)*, pages 166–181, 2009.

[23] O. Udrea, D. Recupero, and V. Subrahmanian. Annotated RDF. In Y. Sure and J. Domingue, editors, *The Semantic Web: Research and Applications*, volume 4011 of *Lecture Notes in Computer Science*, pages 487–501. Springer Berlin Heidelberg, 2006.

[24] O. Udrea, D. R. Recupero, and V. S. Subrahmanian. Annotated RDF. *ACM Trans. Comput. Logic*, 11(2):10:1–10:41, Jan. 2010.

[25] W3C. RDF overview and documentations, 2014. http://www.w3.org/RDF/.

[26] H. Wang, Z. Ma, and J. Cheng. fp-SPARQL: an RDF fuzzy retrieval mechanism supporting user preference. In *Proc. of FSKD*, pages 443–447, 2012.

[27] A. Zimmermann, N. Lopes, A. Polleres, and U. Straccia. A general framework for representing, reasoning and querying with annotated semantic web data. *Web Semant.*, 11:72–95, Mar. 2012.